

ROBOTIZED DETERMINATION OF VITALITY WASTEFULNESS FOR CELL PHONE APPLICATIONS GREED DROID

M. Yasothapriya

Assistant Professor, Department of Computer Science and Applications, Achariya School of Business and Technology/ Manonmaniam Sundaranar University, India.

K. Sadesh

Student, Department of Computer Science and Applications, Achariya School of Business and Technology / Manonmaniam Sundaranar University, India.

ARTICLE INFO

Article History:

Received: 23 Mar 2015;

Received in revised form:

25 Mar 2015;

Accepted: 25 Mar 2015;

Published online: 31 Mar 2015.

Key words:

Android Applications,

Green Droid,

Java Path Finder (JPF)

ABSTRACT

Cell phone applications vitality effectiveness is basic, yet numerous Android applications experience the ill effects of genuine vitality inefficiency issues. Placing these issues is work serious and robotized finding is very alluring. Then again, a key challenge is the absence of a decidable model that encourages robotized judgment of such vitality issues. Our work expects to address this test. We led a top to bottom investigation of 173 open-source and 229 business Android applications, and watched two normal reasons for vitality issues: missing deactivation of sensors or wake bolts, and expense incapable utilization of tangible data. With these discoveries, we propose a computerized way to diagnosing vitality issues in Android applications. Our approach investigates an application's state space by efficiently executing the application utilizing Java Path Finder (JPF). It monitors sensor and wake lock operations to recognize missing deactivation of sensors and wake locks. It likewise tracks the change and utilization of tangible information and judges whether they are successfully used by the application utilizing our state-delicate information utilization metric. Thusly, our methodology can produce point by point reports with significant data to support designers in validating identified vitality issues. We manufactured our methodology as a device, Green Droid, on top of JPF. Actually, we tended to the challenges of producing client cooperation occasions and booking occasion handlers in expanding JPF for breaking down Android applications. We assessed Green Droid utilizing 13 certifiable prevalent Android applications. Green Droid finished vitality productivity finding for these applications shortly. It effectively spotted genuine vitality issues in these applications, and moreover discovered new unreported vitality issues that were later affirmed by designers.

Copyright © 2015 IJASRD. This is an open access article distributed under the Creative Common Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

INTRODUCTION

The cell phone application business sector is becoming quickly. Up until July 2013, the one million Android applications on Google Play store had gotten more than 50 billion downloads^[29]. A large number of these applications influence cell phones' rich peculiarities to give attractive client experiences. For sample, Google Maps can explore clients when they climb in the field by area sensing. Then again, sensing operations are normally vitality consumptive, and constrained battery limit dependably confines such an application's utilization. In that capacity, vitality effectiveness turns into a discriminating sympathy toward cell phone clients. Existing studies demonstrate that numerous Android applications are not vitality productive because of two noteworthy reasons^[54]. In the first place, the Android structure uncovered equipment operation APIs (e.g., APIs for controlling screen brilliance) to developers. Despite the fact that these APIs give adaptability, developers must be in charge of utilizing them carefully since equipment abuse could undoubtedly prompt startlingly expansive vitality waste^[56]. Second, Android applications are basically grown by little groups without committed quality confirmation endeavors. Their engineers once in a while exercise due steadiness in guaranteeing vitality reserve funds. Placing vitality issues in Android applications is troublesome. In the wake of concentrating on 66 genuine bug reports concerning vitality issues, we found that a considerable lot of these issues are discontinuous and just show themselves at certain application states (points of interest are given later in Segment 3). Re-creating these vitality issues is work serious. Developers need to broadly test their applications on different gadgets and perform nifty gritty vitality profiling. To make sense of the underlying drivers of vitality issues, they need to instrument their programs with extra code to log.

Execution follows for determination. Such a procedure is commonly time intensive. This may clarify why a few infamous vitality issues have neglected to be settled in a convenient manner^{[15][40][47]}. In this work, we set out to relieve this trouble by automating the vitality issue determination process. A key exploration challenge for mechanization is the absence of a decidable measure, which permits mechanical judgment of vitality wastefulness issues. All things considered, we began by leading a substantial scale observational study to comprehend how vitality issues have happened in genuine cell phone applications. We researched 173 open-source and 229 commercial Android applications. By analyzing their bug reports, confer logs, bug-altering patches, patch surveys and discharge logs, we mentioned a fascinating objective fact: Although the underlying drivers of vitality issues can fluctuate with distinctive applications, a number of them (more than 60%) are nearly identified with two sorts of hazardous coding phenomena:

Missing Sensor or Wake Lock Deactivation: To utilize a cell phone sensor, an application needs to enlist a listener with the Android OS. The audience ought to be unregistered when the concerned sensor is never again being utilized. Additionally, to make a telephone stay wakeful for calculation, an application needs to procure a wake lock from the Android OS. The procured wake lock ought to additionally be discharged when the calculation finishes. Neglecting to unregister sensor audience members or discharge wake locks could briskly exhaust a completely charged telephone battery^{[5][8]}.

Tangible Information Underutilization: Cell phone sensors test their surroundings and gather tangible information. These information are gotten at high vitality

expense and thusly ought to be used adequately by applications. Poor tangible information usage can likewise bring about vitality waste. For illustration, OSM android, a prominent route application, might continually gather GPS information just to render an imperceptible guide^[51]. This issue happens once in a while at certain application states. Battery vitality is in this way devoured, yet gathered GPS information neglect to deliver any perceptible client advantages. With these discoveries, we propose a way to automatically diagnosing such vitality issues in Android applications. Our methodology investigates an Android application's state space by efficiently executing the application utilizing Java Path Finder (JPF), a broadly utilized model checker for Java programs^[67]. It investigates how tangible information are used at every investigated state, and screening whether sensors/wake locks are appropriately utilized and unregistered/discharged. We have executed this approach as a 18 KLOC augmentation to JPF. The subsequent instrument is named Green Droid. As we will demonstrate in our later evaluation, Green Droid has the capacity investigate the use of location information for the previously stated OSM android application over its 120K states inside three minutes, and effectively find our examined vitality issue. To acknowledge such efficient and compelling investigation, we have to address two re- inquiry issues and two noteworthy specialized issues as takes after.

Research Issues: While existing procedures can be adjusted to screen sensor and wake lock operations to identify their missing deactivation, how to viably identify vitality issues emerging from insufficient employments of sensory information is a remarkable test, which requires promotion dressing two examination issues. To start with, tactile information, once received by an application, would be changed into various structures and utilized by distinctive application parts. Recognizing system information that rely on upon these tactile Information commonly obliges instrumentation of extra code to the first projects. Manual instrumentation is undesirable on the grounds that it is work serious and blunder inclined. Second, regardless of the possibility that a system could be painstakingly instrumented, there is still no decently characterized metric for judging incapable use of tactile information naturally. To address these exploration issues, we propose to screen an application's execution and perform dynamic information stream examination at a byte code guideline level. This permits tactile information utilization to be consistently followed with no requirement for incrementing the concerned projects. We additionally propose a state touchy metric to empower computerized investigation of tactile information use and recognize those application states whose tactile information have been underutilized.

Specialized Issues: JPF was initially intended for analyzing customary Java programs with express control streams^[67]. It executes the byte code of a target Java master gram in its virtual machine. Be that as it may, Android applications are occasion driven and depend enormously on client interactions. Their system code includes numerous approximately coupled occasion handlers, among which no express control stream is determined. At runtime, these occasion handlers are called by the Android structure, which assembles on hundreds of local library classes. In that capacity, applying JPF to investigate Android applications obliges: (1) creating substantial client collaboration occasions, and (2) effectively planning occasion handlers. To address the first specialized issue, we propose to dissect an Android application's GUI design configuration documents, and efficiently count all conceivable client collaboration occasion successions with a limited length at

runtime. We demonstrate that such a limited length does not hinder the viability of our examination, however rather helps rapidly investigate diverse application states and recognize vitality issues. To address the second specialized issue, we introduce an application execution model got from Android particulars. This model catches application-bland transient decides that indicate calling connections between occasion handlers. With this model, we have the capacity to guarantee an Android application to be practiced with right control streams, instead of being arbitrarily planned on its occasion handlers. As we will indicate in our later assessment, the last brings no advantage to the recognizable proof of vitality issues in Android applications in synopsis, we make the accompanying commitments in this article

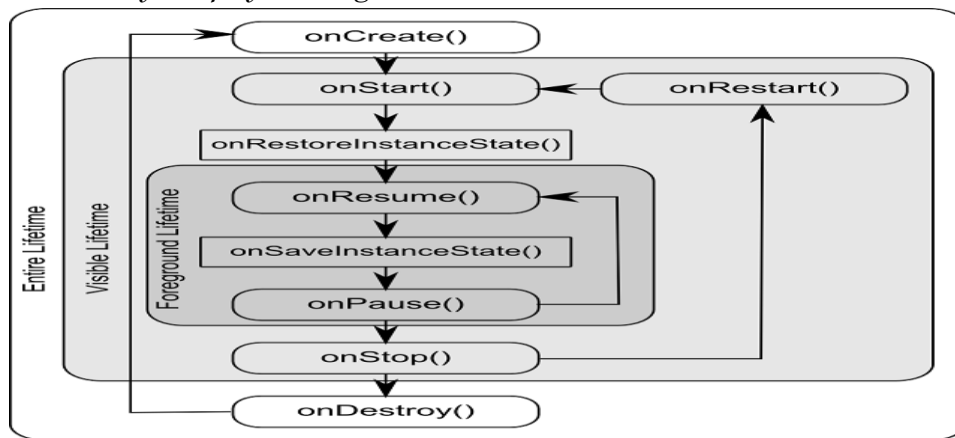
- We exactly ponder genuine vitality issues from 402Android applications. This study recognizes two noteworthy sorts of coding phenomena that ordinarily cause energy issues. We make our exact study information open for exploration purposes^[31].
- We propose a state-based methodology for diagnosing energy issues emerging from tactile information underutilization in Android applications. The methodology systematically investigates an application's state space for such diagnosis reason.
- We exhibit our thoughts for stretching out JPF to examine general Android applications. The examination is in view of an inferred application execution model, which can likewise support other Android application examination errands.
- We execute our methodology as a device, Green Droid, and assess it utilizing 13 certifiable prominent Android applications. Green Droid viably identified 12 genuine vitality issues that had been accounted for, and further found two new vitality issues that were later affirmed by engineers. We were likewise welcomed by engineers to make a patch for one of the two new issues and the patch was acknowledged. These assessment results affirm GreenDroid's adequacy and down to earth helpfulness.

In a preparatory form of this work^[42], we evil presence started the helpfulness of tangible information usage examination in helping designers find vitality issues in Android applications. In this article, we altogether broaden its earlier form in five perspectives: (1) including a far reaching exact investigation of genuine vitality issues gathered from 402 Android applications (Area 3); (2) formalizing the approach of deliberately investigating an Android application's state space for breaking down tactile information utilization (Segment 4.2); (3) improving our tangible information utilization investigation with a result based system, subsequently eliminating human exertion already needed for setting algorithm parameters (Segments 4.4.3 and 6.1); (4) upgrading our assessment with all the more certifiable application subjects, exploration inquiries and result examinations (Segment 5); (5) ex- tending talks of related examination (Area 6).

Whatever is left of this article is composed as takes after. Area 2 presents the nuts and bolts of Android applications. Segment 3 presents our exact study of genuine vitality issues found in Android applications. Segment 4 expounds on our vitality proficiency judgment approach. Area 5 presents our device execution and assesses it with genuine application subjects. Area 6 examines related work, and lastly Segment 7 finishes up this 1: 1K = 1,000 & 1M = 1,000,000; 2: According to Google's classification, there are a total of 32 different categories of Android applications^[28].

Table 1: Project Statistics of Our Studied Android Applications

Application type	Application availability				Application downloads			Covered categories
	Google Code	GitHub	SourceForge	Google Play	Min.	Max.	Avg.	
34 open-source applications with reported energy problems	27/34	8/34	0/34	29/34	1K ¹ ~ 5K	5M ¹ ~ 10M	0.49M ~ 1.68M	15/32 ²
139 open-source applications without reported energy problems	108/139	26/139	10/139	102/139	1K ~ 5K	50M ~ 100M	0.50M ~ 1.22M	24/32
229 commercial applications with energy problems	All are available on Google Play Store				1K ~ 5K	50M ~ 100M	0.77M ~ 2.02M	27/32

Figure 1: An Activity's Lifecycle Diagram

1.1 Foundation:

We select the Android stage for our study in light of the fact that it is at present one of the most broadly received cell phone stages and it is open for examination^[3]. Applications running on Android are essentially written in Java programming dialect. An Android application is initially arranged to Java virtual machine good .class records that contain Java byte code directions. These .class documents are then converted to Davit virtual machine executable .dex records that contain Davit byte code directions. At long last, the .dex records are exemplified into an Android application bundle document (i.e., an .apk record) for circulation and establishment. For simplicity of presentation, we in the taking after may basically allude to "Android application" by "application" when there is no vagueness. An Android application regularly contains four sorts of parts as takes after^[3]: Content suppliers. Content suppliers oversee imparted information components or applications to question or adjust these information. Every application segment will be obliged to take after an endorsed lifecycle that characterizes how this segment is made, utilized, and devastated. Figure 1 demonstrates a movement's lifecycle^[2]. It begins with a call to on Create() handler, and closes with a call to on Destroy() handler. A movement's foreground lifetime begins after a call to on Resume() handler would be called, and the movement would come to forefront once more. In uncommon cases, a stopped or halted action might be executed for discharging memory to different applications with higher needs.

1.2 Issue Extent:

Our chose 173 open-source Android applications contain several bug reports and code corrections. From them, we recognized a sum of 66 bug provides details regarding vitality issues, which cover 34 applications. Among these 66 bug reports, 41 have been affirmed by designers. Most (32/41) affirmed bugs are thought to be not kidding bugs with a seriousness level going from medium to discriminating. Other than that, we discovered 30 of these affirmed bugs have been altered by comparing code amendments, and engineers have checked that these code modifications have in reality tackled relating vitality issues.

Then again, in regards to the 229 business Android applications that experienced vitality issues, we concentrated on their client audits and got three discoveries. To begin with, we found from the audits that several clients complained that these applications drained their cell phone batteries too rapidly and brought about incredible inconvenience for them. Second, as demonstrated in Table 1, these energy issues cover 27 distinctive application classifications, which are very wide when contrasted with the aggregate number of 32 classes. This demonstrates that vitality issues are common to diverse sorts of uses.

Table 1-2

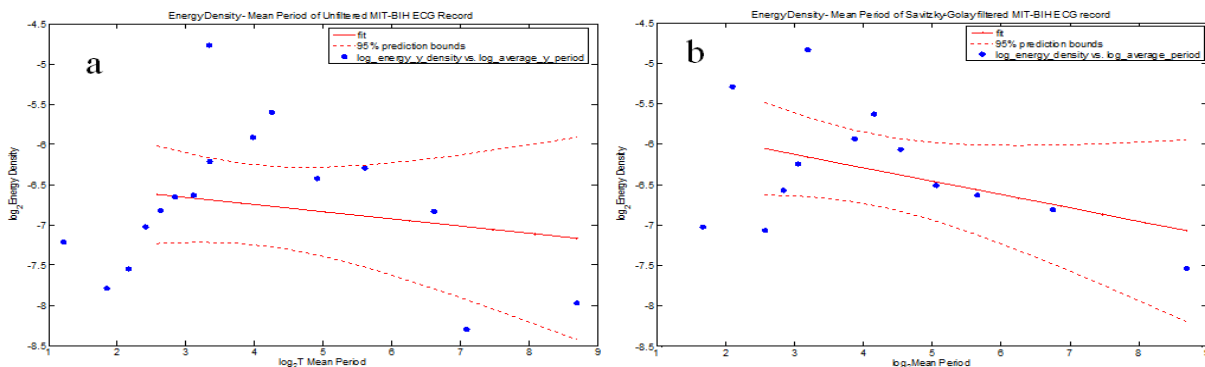


Table 2 rundowns the main five classifications for representation. Third, these 229 commercial applications have gotten more than 176 million down- stacks altogether. This number is huge, and demonstrates that their vitality issues have conceivably influenced an immense number of users. Based on these discoveries, we determine our response to re-inquiry question RQ1: Vitality issues are not kidding. They exist in numerous sorts of Android applications and influence numerous clients. Sides that, we discovered 30 of these affirmed bugs have been settled by relating code corrections, and designers have checked that these code updates have undoubtedly tackled comparing vitality issues.

Then again, in regards to the 229 business Android applications that experienced vitality issues, we contemplated their client surveys and acquired three discoveries. Initially, we found from the audits that several clients complained that these applications drained their cell phone batteries too rapidly and brought on awesome inconvenience for them. Second, as indicated in Table 1, these energy issues cover 27 diverse application classes, which are truly wide when contrasted with the aggregate number of 32 classes. This demonstrates that vitality issues are common to distinctive sorts of uses. Table 2 rundowns the main five classes for delineation. Third, these 229 business applications have gotten more than 176 million down- stacks altogether. This number is huge, and demonstrates that their vitality issues have possibly influenced a boundless number of clients.

In view of these discoveries, we determine our response to re-hunt question RQ1: Vitality issues are not kidding. They exist in numerous sorts of Android applications and influence numerous clients

1.3 Dangers to Legitimacy:

The legitimacy of our exact study may be liable to a few dangers. One is the representativeness of our chose Android applications. To minimize this risk and stay away from subject determination inclination, we chose 173 open-source and 229 business Android applications crossing 27 diverse classes. These applications have been prominently down-stacked and can be great agents of true Android applications. An alternate potential risk is the manual review of our chose subjects. We comprehend that this manual methodology may be blunder inclined. To lessen this danger, we have all our information and discoveries freely reviewed by no less than two scientists. We cross-approved their examination results for consistency.

VITALITY PRODUCTIVITY FINDING

In this area, we expound on our vitality effectiveness diagnosis approach.

2.1 Vitality Utilization Estimation

One noteworthy motivation behind why such a large number of cell phone applications are not vitality productive is that designers need suitable devices to gauge vitality utilization for their applications. Far reaching research has been conveyed to address this topic. Power Tutor^[71] uses system-level power utilization models to gauge the vitality devoured by significant framework segments (e.g., showcase) amid the execution of Android applications. Such models are a capacity of chose framework characteristics (e.g., CPU usage) and obtained by immediate estimations amid the controlling of the gadget's energy state. Sesame^[21] offers the same objective as Power Tutor, however can perform vitality estimation for much littler time interims (e.g., as little as 10ms). E-Prof^[55] is an alternate estimation instrument. As opposed to assessing energy utilization at a framework level like Power Tutor and Sesame, e Prof gauges vitality utilization at an application level by following framework calls made by applications when they run on cell phones. Watts On^[46] further extends reproof's thought by empowering designers to gauge their applications' vitality utilization on their workstations, rather than genuine cell phones. The latest work is eLens^[33]. It consolidates program investigation and every guideline vitality demonstrating to empower much better grained vitality utilization estimation. In any case, eLens expect that cell phone producers ought to give stage subordinate vitality models for every direction. This is not a typical practice as both the equipment and programming of a cell phone stage can advance rapidly. Obliging manufacturers to give another arrangement of guideline level vitality models for every stage redesign is unreasonable. With respect to, eLens gives a equipment based specialized answer for help get such vitality models. Still, power measure equipment may not by and large be available for genuine world engineers. Run of the mill situations for the procedures talked about above are to recognize hotspots (programming segments that consume the most vitality) in cell phone applications, such that engineers can perform vitality utilization optimization. In any case, essentially

knowing the vitality expense of a certain product segment is not satisfactory for an effective improvement assignment. The missing key data is whether this vitality utilization will be vital or not. Consider an application part that constantly uses gathered GPS information to render a guide for route. This part can expend a great deal of vitality and accordingly be identified as a hotspot. Be that as it may, despite the fact that the vitality expense can be high, this part is evitable in that it produces incredible profits for its clients by brilliant route. Accordingly, designers might not need to enhance it. Taking into account this observation, our Green Droid work aides diagnose whether certain vitality devoured by sensing operations can professional duce relating advantages (i.e., high tangible information utilization). This can help engineers settle on astute choices when they confront the decision of whether to improve vitality utilization for certain application segments. For instance, on the off chance that they find that at a few states, sensing operations are performed much of the time, yet therefore gathered sensory information are not adequately used, then they can consider streamlining such sensing components to spare vitality as Geohash Droid designers did^[25]. Have proposed analysis algorithms and automated problem detection in this work, and they have not been covered by these pieces of existing work.

2.2 Data Stream Following:

Dynamic data stream following (DFT for short) observes fascinating information as they proliferate in a program execution^[35]. DFT has numerous helpful applications. For example, Taint Check^[48] utilizes DFT to shield merchandise programming from memory debasement assaults, for example, support floods. It spoils info information from conniving sources and guarantees that they are never utilized as a part of an unsafe way. Taint Droid^[22] keeps Android applications from holing clients' private information. It tracks such information from protection touchy sources, and cautions clients when these information leave the framework. LEAKPOINT^[13] influences DFT to pinpoint memory spills in C and C++ programs. It pollutes dynamically designated memory pieces and screens them on the off chance that their discharge may be overlooked. Our Green Droid work exhibits an alternate use of DFT. We demonstrated that DFT can help track spread of tactile information, such that their usage investigation against vitality utilization can be transmitted to distinguish potential vitality issues in cell phone applications.

CONCLUDING REMARKS

In this article, we introduced an exact investigation of genuine energy issues in 402 Android applications, and recognized two sorts of coding phenomena that regularly cause vitality waste: missing sensor or wake lock deactivation, and tactile information underutilization. In view of these findings, we proposed an approach for robotized vitality issue conclusion in Android applications. Our methodology methodically investigates an application's state space, automatically breaks down its tactile information usage, and monitors the utilization of sensors and wake locks. It aides cheaters place vitality issues in their applications and generates significant reports, which can enormously facilitate the undertaking of duplicating vitality issues and in addition settling them for vitality streamlining. We executed our methodology into an apparatus Green Droid on top of JPF,

and assessed it utilizing 13 certifiable mainstream Android applications. Our experimental results affirmed the viability and commonsense helpfulness of Green Droid.

In future, we plan to study more Android applications and distinguish other regular reasons for vitality issues. For illustration, we discovered from our study that a non-unimportant extent (around 16%) of vitality issues was brought on by system issues (e.g., vitality wasteful information transmission). We are going to study these issues to hide broaden our methodology. Thusly, we expect that our examination will help advance vitality productivity rehearses for a more extensive scope of cell phone applications, and accordingly potentially profit a large number of cell phone clients.

REFERENCES

- [1] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," Proc. ACM SIGSOFT 20th Int'l Symp. Foundations of Soft. Engr. (FSE 12), ACM, 2012, pp. 59:1-59:11.
- [2] "Android Activity Lifecycles." <http://developer.android.com/guide/components/activities.html>.
- [3] "Android Developers Website." <http://developer.android.com/index.html>.
- [4] "Android Emulator." <http://developer.android.com/tools/help/emulator.html>.
- [5] "Android Sensor Management." <http://developer.android.com/reference/android/hardware/SensorManager.html>.
- [6] "Android Platform Versions." <http://developer.android.com/about/dashboards/index.html>.
- [7] "Android Process Lifecycle." <http://reference.android/app/Activity.html#ProcessLifecycle>.
- [8] "Android Power Management." <http://developer.android.com/reference/android/os/PowerManager.html>.
- [9] "And Tweet issue 29." <https://code.google.com/p/and-tweet/issues/detail?id=29>
- [10] M. Arnold, M. Vechev, and E. Yahav, "QVM: an efficient runtime for detecting defects in deployed systems," ACM Trans. Software Engineering and Methodology, vol. 21, 2011, pp. 2:1-2:35.
- [11] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps," Proc. ACM SIGPLAN Int'l Conf. Object-oriented Programming Systems Languages & Applications (OOPSLA 13), ACM, pp. 641-660.
- [12] J. Clause, W. Li, and A. Orso, "Dytan: a generic dynamic taint analysis framework," Proc. Int'l Symp. Software Testing and Analysis (ISSTA07), 2007, pp. 196-206.
- [13] J. Clause and A. Orso, "LEAKPOINT: pinpointing the causes of memory leaks," Proc. Int'l Conf. Soft. Engr. (ICSE 10), pp. 515-524.
- [14] "Crawler4j." <https://code.google.com/p/crawler4j/>.
- [15] "CSipSimple issue 81." <https://code.google.com/p/csipsimple/issues/detail?id=81>.
- [16] "CSipSimple issue 744." <https://code.google.com/p/csipsimple/issues/detail?id=744>.
- [17] "CSipSimple issue 1674." <https://code.google.com/p/csipsimple/issues/detail?id=1674>.
- [18] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," Proc. Int'l Conf. Mobile Systems, Applications, and Services (MobiSys 10), ACM, 2010, pp. 49-62.
- [19] "dex2jar." <https://code.google.com/p/dex2jar/>.
- [20] I. Dillig, T. Dillig, E. Yahav, & S. Chandra, "The CLOSER: automating resource management in Java," Proc. Int'l Symp. Memory Management (ISMM 08), ACM, 2008, pp. 1-10.

- [21] M. Dong and L. Zhong, "Sesame: Self-constructive high-rate system energy modeling for battery-powered mobile systems," Proc. Int'l Conf. Mobile Systems, Applications, and Services (Mobisys 11), ACM, 2011, pp. 335-348. Eng. Notes, vol. 37, 2012, pp. 1-5.
- [22] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A.N. Sheth, "Taint Droid: An information-flow tracking system for real-time privacy monitoring on smartphones," Proc. USENIX Conf. Operating Systems Design and Impl. (OSDI 10), 2010, pp. 393- 407.
- [23] K. Etessami, and T. Wilke, "An Until hierarchy for temporal logic," Proc. IEEE Symp. Logic in Computer Science (LICS 96). 1996, pp. 108-117.
- [24] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android per- missions demystified," Proc. ACM Conf. Computer and Communications Security (CCS 11), 2011, pp. 627-638.
- [25] "GeoHashDroid issue 24." <https://code.google.com/p/geohashdroid/issues/detail?id=24>
- [26] "Google Code website." <https://code.google.com/>
- [27] "GitHub website." <https://github.com/>
- [28] "Google Play store website." <https://play.google.com/store> .
- [29] "Google Play Wiki Page." http://en.wikipedia.org/wiki/Google_Play .
- [30] "GPSLogger issue 7." <https://code.google.com/p/gpslogger/issues/detail?id=7>
- [31] "GreenDroid project." <https://sccpu2.cse.ust.hk/greendroid> (id: greendroid, password: tsesubmission).
- [32] C. Guo, J. Zhang, J. Yan, Z. Zhang, and Y. Zhang, "Characterizing and detecting resource leaks in Android applications," Proc. ACM/IEEE Int'l Conf. Automated Soft. Engr. (ASE 13), 2013, pp. 389-398.
- [33] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," Proc. 35th Int'l Conf. Soft. Engr. (ICSE 13), pp. 92-101.
- [34] C. S. Jensen, M. R. Prasad, and A. Møller, "Automated testing with targeted event sequence generation," Proc. Int'l Symp. Software Testing and Analysis (ISSTA 13), 2013, pp. 67-77.
- [35] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis, "libdft: practical dynamic data flow tracking for commodity systems," Proc. ACM Conf. Virtual Exe. Env. (VEE 12), 2012, pp. 121-132.
- [36] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," Proc. Int'l Conf. Mobile Sys., App's, and Services (MobiSys 08), 2008, pp. 239-252.
- [37] D. King, B. Hicks, M. Hicks, and T. Jaeger, "Implicit Flows: Can't Live with `Em, Can't Live without `Em," Proc. Int'l Conf. Information Systems Security (ICISS 08), 2008, pp. 56-70.
- [38] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær, "EnTracked: energy-efficient robust position tracking for mobile devices," Proc. Int'l Conf. Mobile Systems, Applications, and Services (MobiSys 09), ACM, 2009, pp. 221-234.
- [39] "K9Mail issue 1986." <https://code.google.com/p/k9mail/issues/detail?id=1986>
- [40] "K9Mail issue 3348." <https://code.google.com/p/k9mail/issues/detail?id=3348>
- [41] Y. Liu, C. Xu and S.C. Cheung, "Verifying Android applications Using Java Pathfinder," Technical Report HKUST-CS-12-03.
- [42] Y. Liu, C. Xu and S.C. Cheung, "Where has my battery gone? Finding sensor related energy black holes in smartphone applications," Proc.11th IEEE Int'l Conf. Pervasive Computing and Communication (PERCOM 13), 2013, pp. 2-10.

- [43] X. Ma, P. Huang, X. Jin, P. Wang, S. Park, D. Shen, Y. Zhou, L. K. Saul, and G. M. Voelker, "eDoctor: Automatically diagnosing abnormal battery drain issues on smartphones," Proc. 10th ACM/USENIX Symp. Networked Systems Design and Implementation (NSDI 13), April 2013, pp. 57-70.
- [44] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," The annals of mathematical statistics, vol. 18, 1947, pp. 50-60.
- [45] N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani, and R. Mahmood, "Testing android apps through symbolic execution," SIGSOFT Softw
- [46] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," Proc. 18th Int'l Conf. Mobile Computing and Networking (Mobicom 12), 2012, pp. 317 - 328.
- [47] "My Tracks issue 520." <https://code.google.com/p/mytracks/issues/detail?id=520>
- [48] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," Proc. ISOC Network and Distributed System Security Symp., 2005.
- [49] D. Ocateau, S. Jha, and P. McDaniel, "Retargeting Android applications to Java bytecode," Proc. ACM SIGSOFT Int'l Symp. Foundations of Soft. Engr. (FSE 12), ACM, 2012, pp. 6:1-6:11.
- [50] "Omniroid issue 179." <https://code.google.com/p/omniroid/issues/detail?id=179>
- [51] "Osmroid issue 53." Available: <https://code.google.com/>
- [52] S. Park, B. M. M. Hossain, I. Hussain, C. Csallner, M. Grechanik, K. Taneja, C. Fu, and Q. Xie, "CarFast: achieving higher statement coverage faster," Proc. ACM SIGSOFT Int'l Symp. Foundations of Soft. Engr. (FSE 12), ACM, 2012, pp. 35:1-35:11.
- [53] C. S. Păsăreanu, P. C. Mehrlitz, D. H. Bushnell, K. Gundy-Burlet, M. Lowry, S. Person, and M. Pape, "Combining unit-level symbolic execution and system-level concrete execution for testing NASA Software," Proc. Int'l Symp. Software Testing and Analysis (ISSTA 08), ACM, 2008, pp. 15-26.
- [54] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," Proc. ACM Workshop on Hot Topics in Networks (Hotnets 11), ACM, 2011, pp. 5:1-5:6.
- [55] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? Fine grained energy accounting on smartphones with Eprof," Proc. Euro. Conf. Comp. Sys. (EuroSys 12), 2012, pp. 29-42.
- [56] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake? Characterizing and detecting no-sleep energy bugs in smartphone apps," Proc. 10th Int'l Conf. Mobile Systems, Applications, and Services (MobiSys 12), 2012, pp. 267-280.
- [57] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive GPS-based positioning for smartphones," Proc. Int'l Conf. Mobile Systems, App., and Services (MobiSys 10), ACM, 2010, pp. 299 - 314.
- [58] B. Priyantha, D. Lymberopoulos, and J. Liu, "LittleRock: Enabling Energy-Efficient Continuous Sensing on Mobile Phones," IEEE Pervasive Computing, vol. 10, 2011, pp. 12-15.
- [59] M. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," Proc. Int'l Conf. Mobile Systems, Applications, and

- Services (MobiSys 10), ACM, 2010, pp. 255-270.
- [60] “Real APKLeecher.” <https://code.google.com/p/real-apk-leecher>
- [61] “Robotium, a testing framework for Android applications.” <http://code.google.com/p/robotium/>.
- [62] E. J. Schwartz, T. Avgerinos, and D. Brumley, “All you ever wanted to know about dynamic taint analysis and forward symbolic execution,” Proc. IEEE Symp. Security and Privacy, 2010, pp. 317-331.
- [63] “SourceForge website.” URL: <https://sourceforge.net/>
- [64] “Sofia Public Transport Nav issue 38.” <https://code.google.com/p/sofia-public-transport-navigator/issues/detail?id=38>
- [65] “Sofia Public Transport Nav issue 76.” <https://code.google.com/p/sofia-public-transport-navigator/issues/detail?id=76>
- [66] E. Torlak and S. Chandra, “Effective interprocedural resource leak detection,” Proc. Int’l Conf. Soft. Engr. (ICSE 10), 2010, pp. 535-544.
- [67] W. Visser, K. Havelund, G. Brat, and S. Park, “Model checking programs,” Proc. Int’l Conf. Automated Soft. Engr., 2000, pp. 3-11.
- [68] W. Weimer and G. C. Necula, “Finding and preventing run-time error handling mistakes,” Proc. ACM SIGPLAN Conf. Object-oriented Prog., Sys, Lang., and App’s (OOPSLA 04), 2004, pp. 419-431.
- [69] W. Yang, M. R. Prasad, and T. Xie, “A grey-box approach for automated GUI-model generation of mobile applications,” Lecture Notes in Computer Science, vol. 7793, 2013, pp. 250-265.
- [70] L. Zhang, M. S. Gordon, R. P. Dick, Z. M. Mao, P. Dinda, and L. Yang, “ADEL: an automatic detector of energy leaks for smartphone applications,” Proc. 8th IEEE/ACM/IFIP Int’l Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS 12), 2012, pp. 363-372.
- [71] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. M. Mao, and L. Yang, “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” Proc. Int’l Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS 10), 2010, pp. 105-114.
- [72] “Zmanim issue 50/56.” <https://code.google.com/p/android-zmanim/issues/detail?id=50/56>